Interactive Video Sculpture

Let's get creative with media in p5.js! In this project, we'll use p5.js to create an interactive video sculpture inspired by the artist Nam June Paik.

Nam June Paik (1932 - 2006) was a Korean American artist considered to be the founder of video art. Paik is well known for his video sculptures consisting of found television monitors, where he experimented with ideas about video transmission and manipulation—across an era when video and TV were emerging technologies.

Together, we'll load and draw videos to create a "sculpture" inspired by the late artist's work. We don't have extra TV monitors to spare, but we can simulate them on the canvas using p5.js!

Tasks

18/18 complete
Mark the tasks as complete by checking them off

Prepare the Videos

1.
Take a look at the p5.js sketch to the right. It contains a simple grid of "TV screens", made using the `rect()` function over a nested `for` loop that calculates the eventual locations of our videos. Our job is to fill these screens in with videos!

First, we'll need to load the videos. Take a look at the video paths stored in `cloudPath`, `starsPath`, `staticPath`, and `humanPath`. In the `setup()` function, use the `createVideo()` function to load these videos into their respective variables: `cloudVideo`, `starsVideo`, `staticVideo`, and `humanVideo`.

Click "Save" to run your code to see the first frames of the four videos appear behind the grid of TV screens!
Stuck? Get a hint

2.
This is quite a lot of videos to work with! Let's make them easier to manage.

After loading all of these videos, place all of the newly created video elements in a single array stored in the `videos` variable.
Stuck? Get a hint

3.
Let's get our videos playing! They are all pretty short, so let's make them loop to run the video sculpture continuously.

Still inside the `setup()` function and using the `videos` array we just populated, iterate over each video element and apply the `.loop()` method to each video.

After correctly calling the `.loop()` method, you may notice that each video is still not playing—that's OK, we'll address that next.
Stuck? Get a hint
4.
Even though we called the `.loop()` method, you may notice the videos are still not playing! If you're experiencing this strange behavior, it's due to web security reasons. By default, many web browsers block videos from auto-playing on websites—except in certain cases, like if the audio is muted or the user interacts with the site by clicking, pressing keys, etc.

This makes sense—imagine how annoying it would be if websites could play whatever loud videos they wanted to when we least expect it! But it can also make creative, video-based projects a bit more complicated. To work around this issue and get our videos playing automatically, we'll need to mute each video before attempting to play it.

While iterating over all the videos, use the `.volume()` method to set each video's volume level to 0. Make sure to do this before calling the `.loop()` method.

Run your code to see the videos playing behind the grid of TV screens!
Stuck? Get a hint
5.
You'll also notice that our videos are also currently showing up as HTML elements in the background of the canvas. We don't want those in our final sketch, so let's get rid of them.

While iterating over all the videos, apply `.hide()` method to each one.

Run your code to see the HTML videos behind the TV screens disappear.
Stuck? Get a hint
6.
Finally, let's populate another array inside the `outsideVideos` variable. This is a bit different than the `videos` array—inside `outsideVideos`, we'll store

three of our four loaded videos, to later draw on the outside, or perimeter, of the grid of “screens”. The fourth video, `humanVideo`, will later be placed in the remaining four “screens” in the center of the grid.

Store an array containing `cloudVideo`, `starsVideo`, and `staticVideo` inside the `outsideVideos` variable.
Stuck? Get a hint

Draw the Inside Videos

7.
Now our videos are prepped and ready to be used in the canvas!

Take a deeper look at the code within the `draw()` function. You’ll see it uses a nested `for` loop, iterating over the dimensions of our grid. These dimensions are given by the
variables `numOfScreensWide` and `numOfScreensTall`.

Inside the loops, we’ll use the `i` and `j` variables to know which screen we draw to the canvas. To help you visualize which screen is at
what `i`,`j` position, refer to this diagram below.

The 4 inside screens, which we’ll be working with first, are color-coded yellow. We’ll draw `humanVideo` into these 4 screens. The entire video will be divided across them, with each screen holding the respective top-left, top-right, bottom-left, or bottom-right quadrant of the video.

We’ll begin with the screen at `i === 1` and `j === 1`, which will eventually hold the top-left quadrant of `humanVideo`.

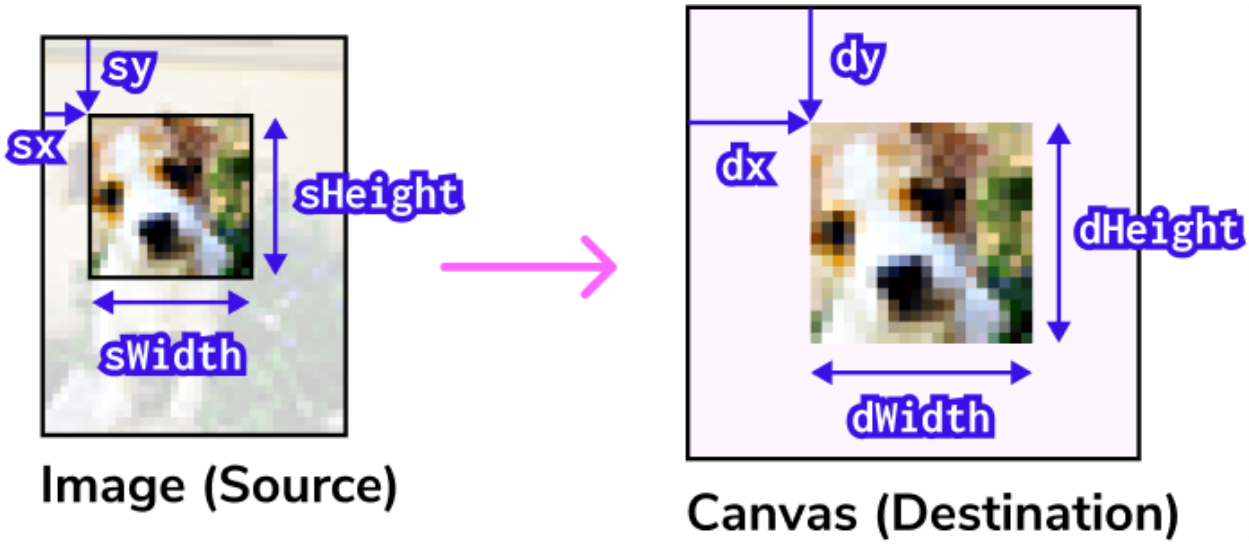
To do this, first create an `if` statement that checks whether we are currently at the screen `i === 1`, `j === 1`. You should create this logic after the white rectangle is drawn, within the nested `for` loop.
Stuck? Get a hint

8.
Next, we’ll draw the top-left quadrant of `humanVideo` to this screen! We’ll do this by using the `image()` function in a new way.

We've seen how to supplying the `image()` function with up to 5 arguments: the image or video asset to draw, x and y positions, and a width and height. The `image()` function can also take in 4 additional arguments, allowing us to draw cropped regions of an image or video to the canvas. Take a look at the syntax below:

```
image(img, dx, dy, dWidth, dHeight, sx, sy, sWidth, sHeight);
```

We're already familiar with the arguments beginning with 'd' (or destination). However, the arguments starting with 's' (or source) describe the rectangular region of the source material you want to draw. This region is determined by a starting position, a width, and a height. To get a better idea, view the diagram below:



With this in mind, use the `image()` function inside the `if` statement you just created. Add arguments that draw the top-left quadrant of `humanVideo` to the screen at `i === 1`, `j === 1`.

Run the sketch to see that only the video's top-left area draws to the screen!
Stuck? Get a hint
9.
Now that we have a taste for how that works, let's draw another screen!

Next, we'll work with the screen at `i === 1`, `j === 2`. This is the screen directly below the one we just drew a video to.

Inside the nested `for` loop, create another `if` statement that now checks whether the current screen corresponds with `i === 1` and `j === 2`.

Then, applying your knowledge of the `image()` function's optional parameters, use the `image()` function to draw the bottom-left quadrant of `humanVideo` to this screen.

Run the code to see the bottom-left area of the video drawn to the screen!
Stuck? Get a hint

10.

Next, we'll work with the screen at `i === 2` and `j === 1`, which is directly to the right of the first one we worked with.

Inside the nested `for` loop, create another `if` statement that now checks whether the current screen corresponds with `i === 2`, `j === 1`.

Then, use the `image()` function and its optional parameters to draw the top-right quadrant of `humanVideo` to this screen.

Run the code to see the top-right area of the video drawn to the screen!
Stuck? Get a hint

11.

Next, we'll work with the screen at `i === 2`, `j === 2`. This is the final screen for `humanVideo`, and is below the one we just worked with in task 10.

Inside the nested `for` loop, create another `if` statement that now checks whether the current screen corresponds with `i === 2` and `j === 2`.

Then, use the `image()` function and its optional parameters to draw the bottom-right quadrant of `humanVideo` to this screen.

Run the code to see all four areas of the `humanVideo` video drawn to the screen!
Stuck? Get a hint

Draw the Outside Videos

12.

Now let's fill the screens on the outside of the grid!

We need to check if the current screen we're drawing to is on the outside of the grid. You can do this in multiple ways.

One simple way is to record the `i` and `j` values for each of these outside screens, and check if any of them match. A more elegant way is to

structure your previous logic that draws the inner screens, so that if the `i` and `j` values do not match the `if` statement conditions for the inner videos, they default to the code written in the final `else` statement.

Use one of these approaches to create this logical condition. We'll work inside it in the next tasks!
Stuck? Get a hint

13.

Now that we checked for if we're at an outside screen, let's find a way to pick a video to draw at this location.

Nam June Paik's video sculptures are known to have a dynamic and random quality—so let's create a way to generatively fill in these screens!

We want to select a video in our collection of videos to place on this screen, but we can't use the `random()` function. If we did, it would change which video shows up every time the `draw()` loop iterates, which would look too chaotic!

Instead, we need to find some way to translate the current `i` and `j` position to a specific video in our collection, in a way that doesn't change when the next iteration of `draw()` is called. Inside the logic you just created that checks if we are at a screen on the outer grid, use the formula below:

```
(i + j) % outsideVideos.length
```

and set it to a new variable called `selectedIndex`. This value represents an index value we'll soon use to access a video element within the `outsideVideos` array.

To break this strange formula down, the `(i + j)` part lets us translate the current screen at position `i` and `j` into some new value. The `%`, or remainder operator, allows us to translate `(i + j)` to some index that is within the bounds of the `outsideVideos` array.
Stuck? Get a hint

14.

Using the value you just calculated in `selectedIndex`, select the video element in the `outsideVideos` array with that index. Set the selected video element to a new variable called `selectedVideo`.
Stuck? Get a hint

15.
Draw the video you just selected! Use the `image()` function to draw the chosen video to the canvas, positioning and scaling it to the size of the existing white rectangle. This time, instead of drawing a cropped portion of the video, you'll just draw the entire video.

Run the sketch—you should now see all of the screens playing videos!
Stuck? Get a hint

Add Interactivity

16.
Finally, let's make this sketch more interesting with interactivity! Inspired by the dynamic, changing TV displays in Paik's video art, we'll make the outside videos change when the mouse is pressed.

To do this, first, add a `mouseClicked()` function into your sketch.
Stuck? Get a hint

17.
Before we write anything in the `mouseClicked()` function, take a look at the `counter` variable defined towards the beginning of **sketch.js**. We will use this variable to eventually make mouse clicks trigger a change in which videos are displayed in the outside of the grid.

Inside the line where you calculated the selected index for an outside video, change the formula to:

```
(i + j * counter) % outsideVideos.length
```

This formula is slightly arbitrary—the main difference is that the `counter` variable now affects which video is selected.
Stuck? Get a hint

18.
Finally, let's hook it all together! Inside the `mouseClicked()` function, write code that increments the `counter` variable when the mouse is clicked.

When you're done, run the sketch to see the videos on the outside of the grid change when you click anywhere on the canvas!

As an optional next step, try playing around with the formula in the previous step. What happens when you combine `i`, `j`, and `counter` in different ways within the formula, using different arithmetic operators?

How does this affect which videos are chosen, and how they change on click?

Also, feel free to play around with other formats for interactivity! What if clicking on an individual screen changed what video plays on it? What if hovering over a screen changed its video?

Hint

There are several ways to increment a numeric variable. You can use the following syntax:

```
num++;
```

which is the same as writing:

```
num = num + 1;
```

sketch.js

```
let cloudPath = 'https://static-assets.codecademy.com/Courses/Learn-p5/projects/cloud.mp4';
let starsPath = 'https://static-assets.codecademy.com/Courses/Learn-p5/projects/stars.mp4';
let staticPath = 'https://static-assets.codecademy.com/Courses/Learn-p5/projects/static.mp4';
let humanPath = 'https://static-assets.codecademy.com/Courses/Learn-p5/projects/human.mp4';

let cloudVideo, starsVideo, staticVideo, humanVideo;
let videos;
let outsideVideos;

let margin = 20;
let numOfScreensTall = 4;
let numOfScreensWide = 4;

let counter = 1;

function setup() {
  createCanvas(600, 500);
  // TODO: Load videos
  cloudVideo = createVideo(cloudPath);
  starsVideo = createVideo(starsPath);
```

```
  staticVideo = createVideo(staticPath);
  humanVideo = createVideo(humanPath);
  // TODO: Populate videos array
  videos = [cloudVideo, starsVideo, staticVideo, humanVideo];
  // TODO: Iterate over videos to loop, mute, and hide each one
  for (let i = 0; i < videos.length; i++) {
    let video = videos[i];
    video.volume(0);
    video.loop();
    video.hide();
  }
  // TODO: Populate outsideVideos array
  outsideVideos = [cloudVideo, starsVideo, staticVideo];
}

function draw() {
  background(0);

  // Calculate the width and height for each "screen" in the grid
  let w = (width - margin * (numOfScreensWide + 1)) / numOfScreensWide;
  let h = (height - margin * (numOfScreensWide + 1)) / numOfScreensWide;

  // Create a 4x4 grid of screens with a margin of 20px
  for (let i = 0; i < numOfScreensWide; i++) {
    for (let j = 0; j < numOfScreensTall; j++) {

      // Calculate current x, y position where this "screen" should be drawn
      let x = margin + i * (w + margin);
      let y = margin + j * (h + margin);

      // Draw a white rectangle to demonstrate where this "screen" is
      fill(255);
      rect(x, y, w, h);

      // TODO: Fill this "screen" with a video, according to its (i,j) position
      if (i === 1 && j === 1) {
        image(humanVideo, x, y, w, h, 0, 0, humanVideo.width/2, humanVideo.height/2);
      }
      else if (i === 1 && j === 2) {
```

```
        image(humanVideo, x, y, w, h, 0, humanVideo.height/2, humanVideo.width/2,
humanVideo.height/2);
      }
      else if (i === 2 && j === 1) {
        image(humanVideo, x, y, w, h, humanVideo.width/2, 0, humanVideo.width/2,
humanVideo.height/2);
      }
      else if (i === 2 && j === 2) {
        image(humanVideo, x, y, w, h, humanVideo.width/2, humanVideo.height/2,
humanVideo.width/2, humanVideo.height/2);
      } else {
          let selectedIndex = (i + j * counter) % outsideVideos.length;
          let selectedVideo = outsideVideos[selectedIndex];
          image(selectedVideo, x, y, w, h);
      }
    }
  }
}

// TODO: Make videos on the outside change when mouse is clicked
function mouseClicked() {
  counter++;
}
```